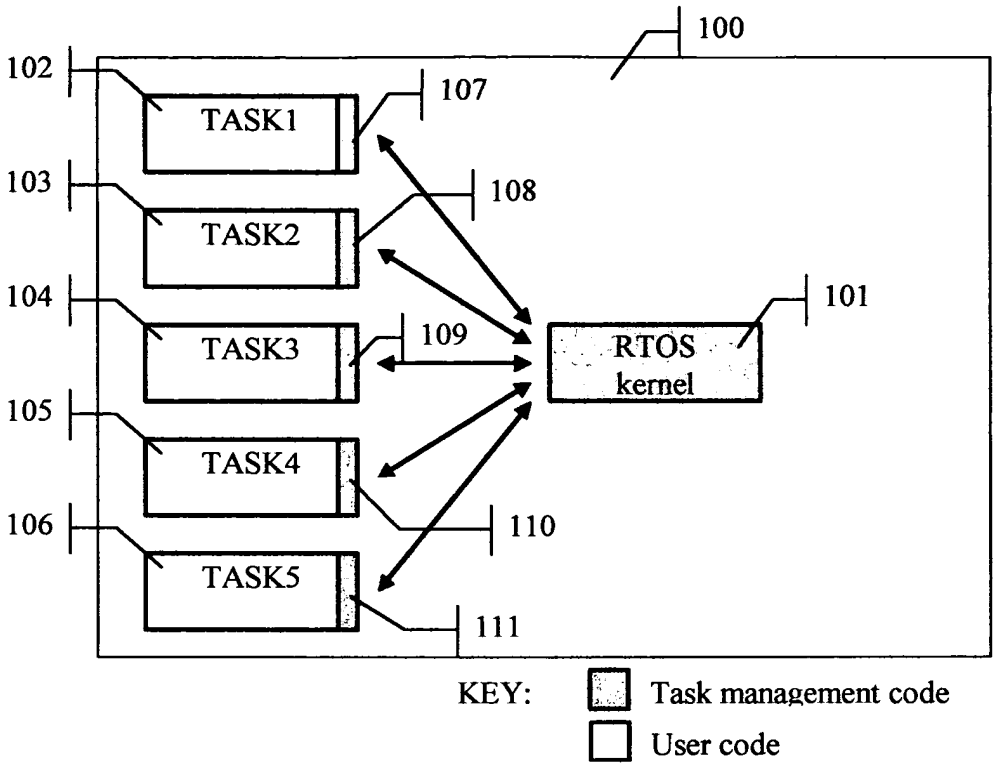
100
102
TASK1
107
103
TASK2
108
104
TASK3
109
101
RTOS
kernel
105
TASK4
110
106
TASK5
111
KEY:
Task management code
User code

Figure 1

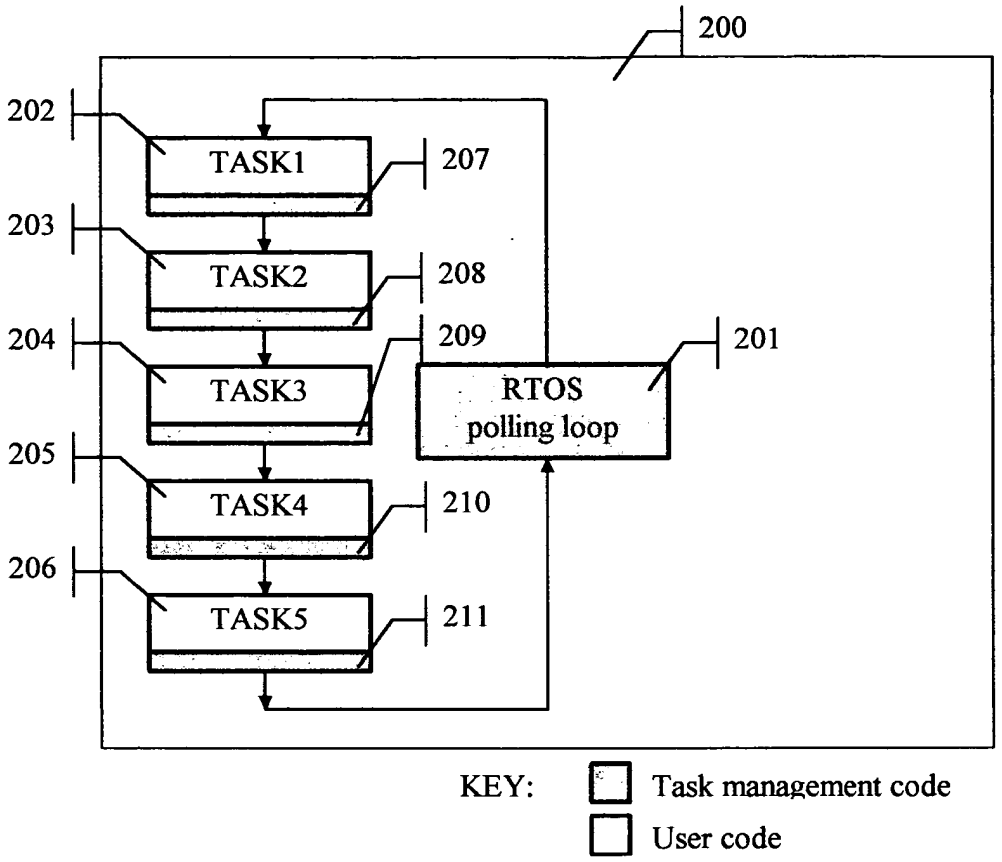
200
202
TASK1
207
203
TASK2
208
204
209
201
TASK3
RTOS
polling loop
205
TASK4
210
206
TASK5
211
KEY:
Task management code
User code

Figure 2

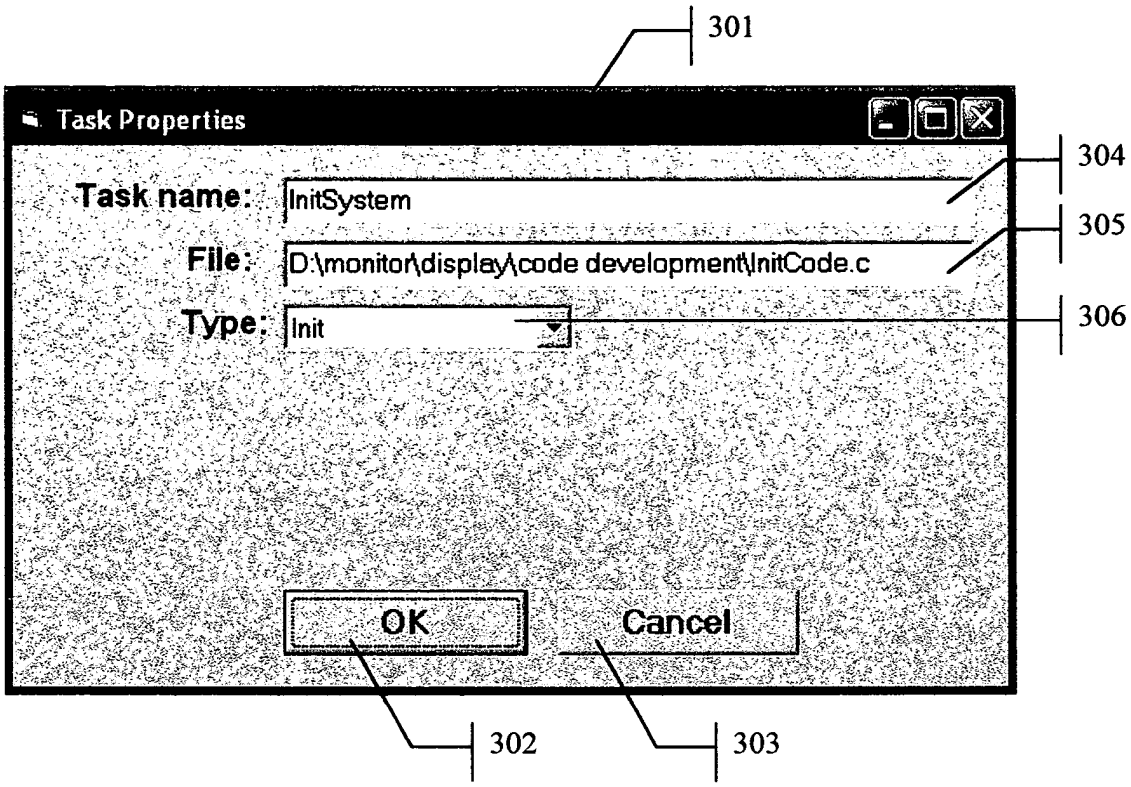
301
Task Properties
Task name:
InitSystem
File:
D:\monitor\display\code development\InitCode.c
Type:
Init
OK
Cancel
304
305
306
302
303

Figure 3

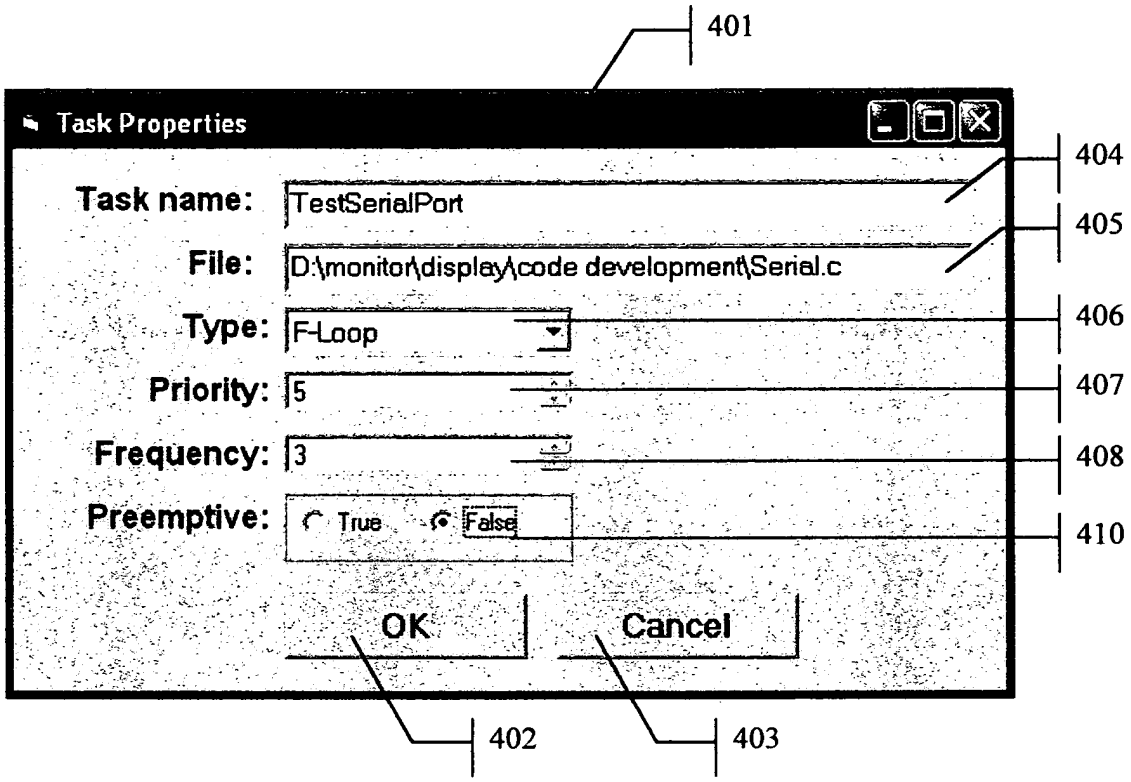
401
Task Properties
Task name:
TestSerialPort
File:
D:\monitor\display\code development\Serial.c
Type:
F-Loop
Priority:
5
Frequency:
3
Preemptive:
True
False
OK
Cancel
402
403
404
405
406
407
408
410

Figure 4

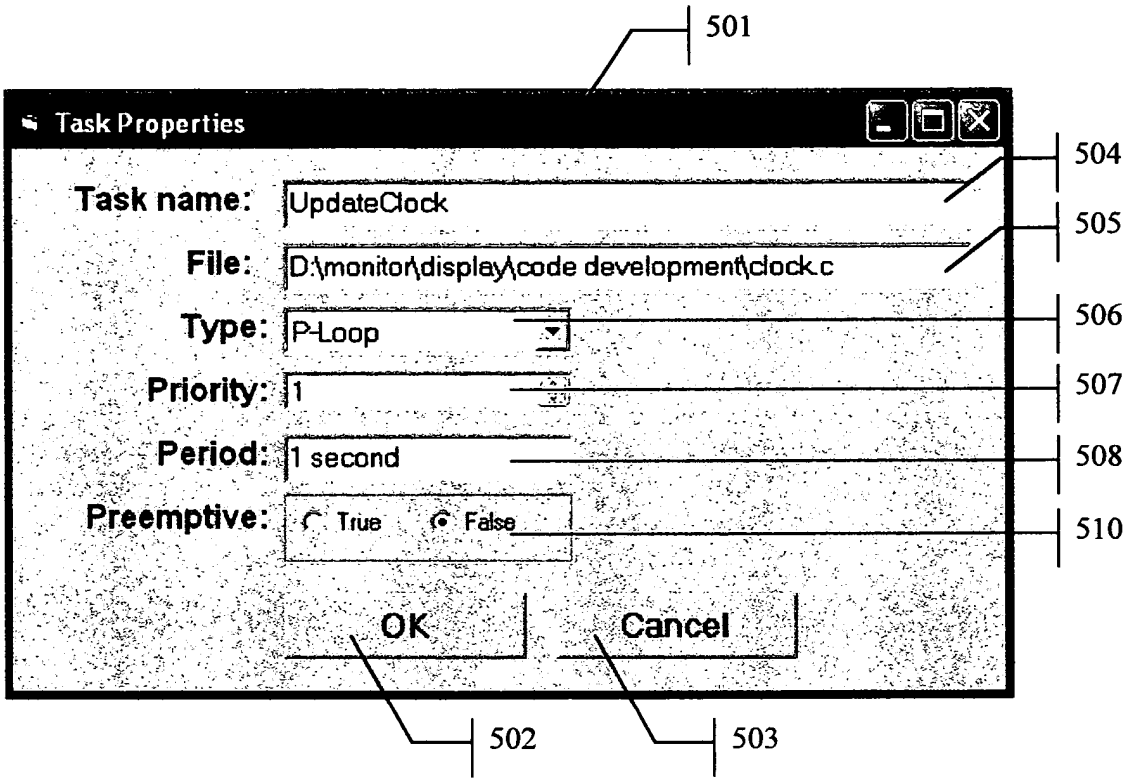
501
Task Properties
Task name:
UpdateClock
File:
D:\monitor\display\code development\clock.c
Type:
P-Loop
Priority:
1
Period:
1 second
Preemptive:
True
False
OK
Cancel
502
503
504
505
506
507
508
510

Figure 5

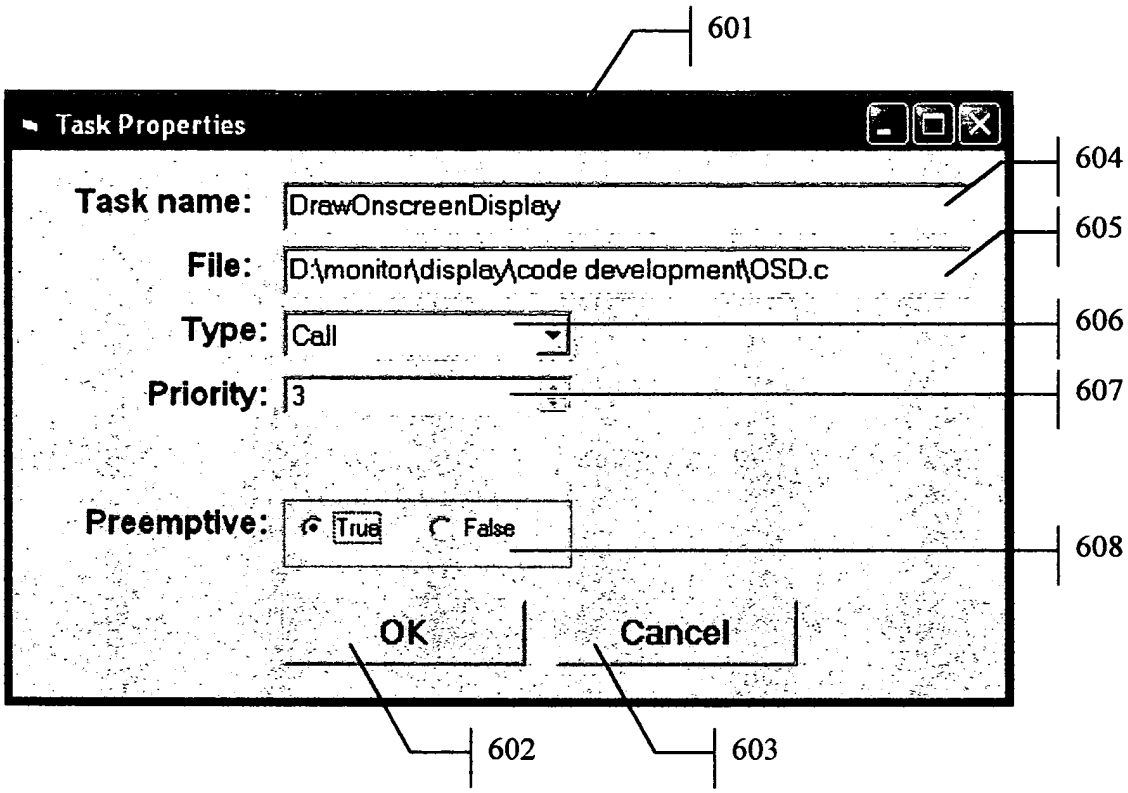
601
Task Properties
Task name:
DrawOnscreenDisplay
File:
D:\monitor\display\code development\OSD.c
Type:
Call
Priority:
3
Preemptive:
True
False
OK
Cancel
604
605
606
607
608
602
603

Figure 6

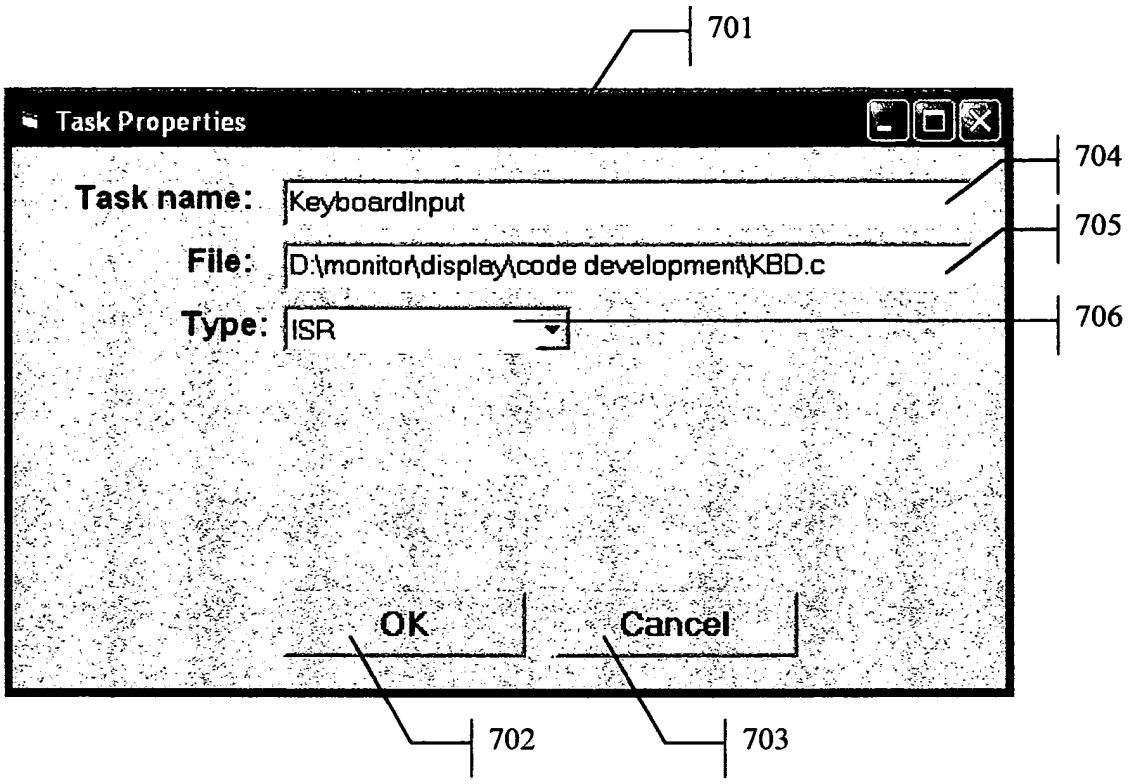
701
Task Properties
Task name:
KeyboardInput
File:
D:\monitor\display\code development\KBD.c
Type:
ISR
OK
Cancel
704
705
706
702
703

Figure 7

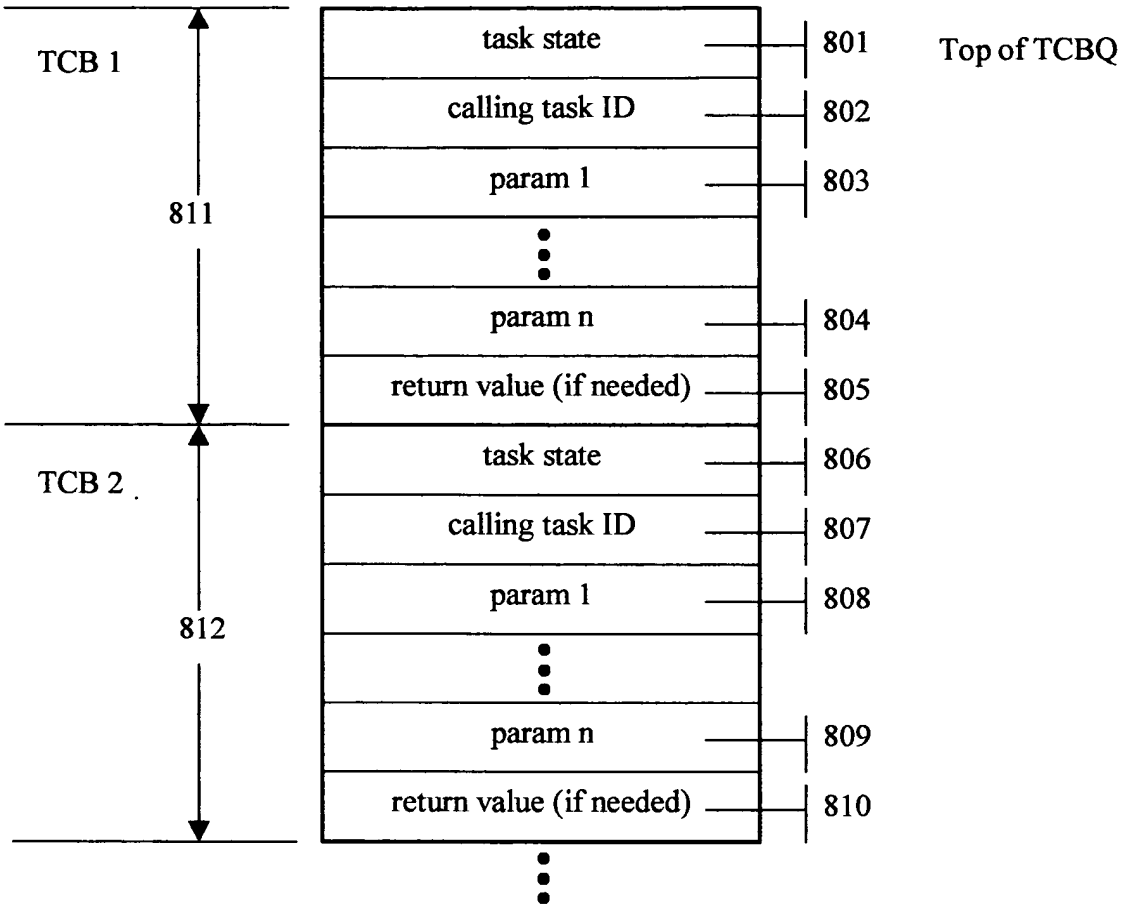

Figure 8

`SynthOS_X_write(SynthOS_taskx_TCBQ, n, x);`	Writes value x to n positions below top of TCBQ for task x
`SynthOS_X_write_next(SynthOS_taskx_TCBQ, x);`	Writes value x to next empty position in TCBQ for task x
`SynthOS_X_read(SynthOS_taskx_TCBQ, n, x);`	Reads value x from n positions below top of TCBQ for task x
`SynthOS_X_discard(SynthOS_taskx_TCBQ, n);`	Pops n locations off top of TCBQ, discards values popped, writes first location with 0

Figure 9

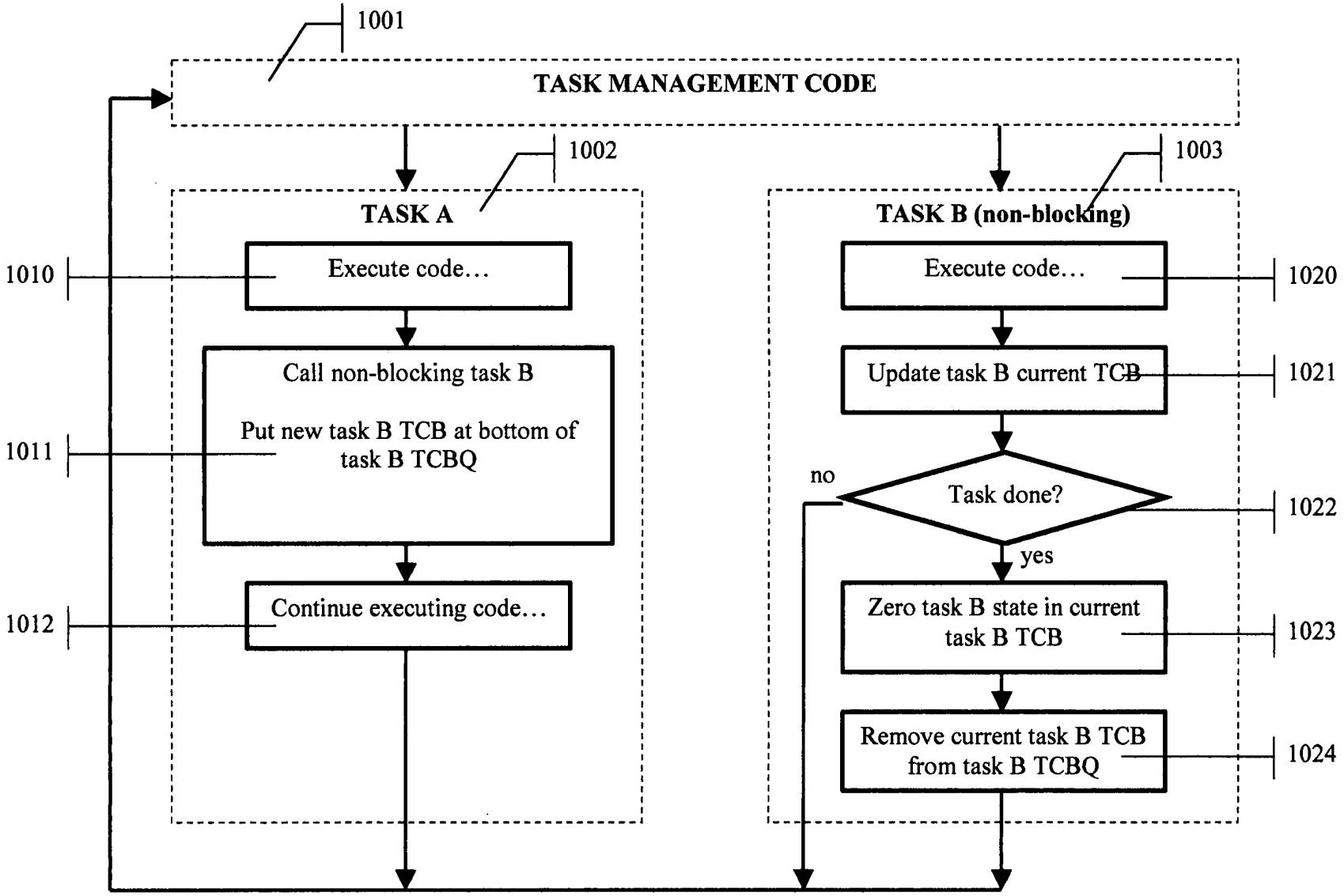
1001
TASK MANAGEMENT CODE
1002
TASK A
1003
TASK B (non-blocking)
1010
Execute code...
Call non-blocking task B
Put new task B TCB at bottom of task B TCBQ
1011
Continue executing code...
1012
Execute code...
1020
Update task B current TCB
1021
no
Task done?
1022
yes
Zero task B state in current task B TCB
1023
Remove current task B TCB from task B TCBQ
1024

Figure 10

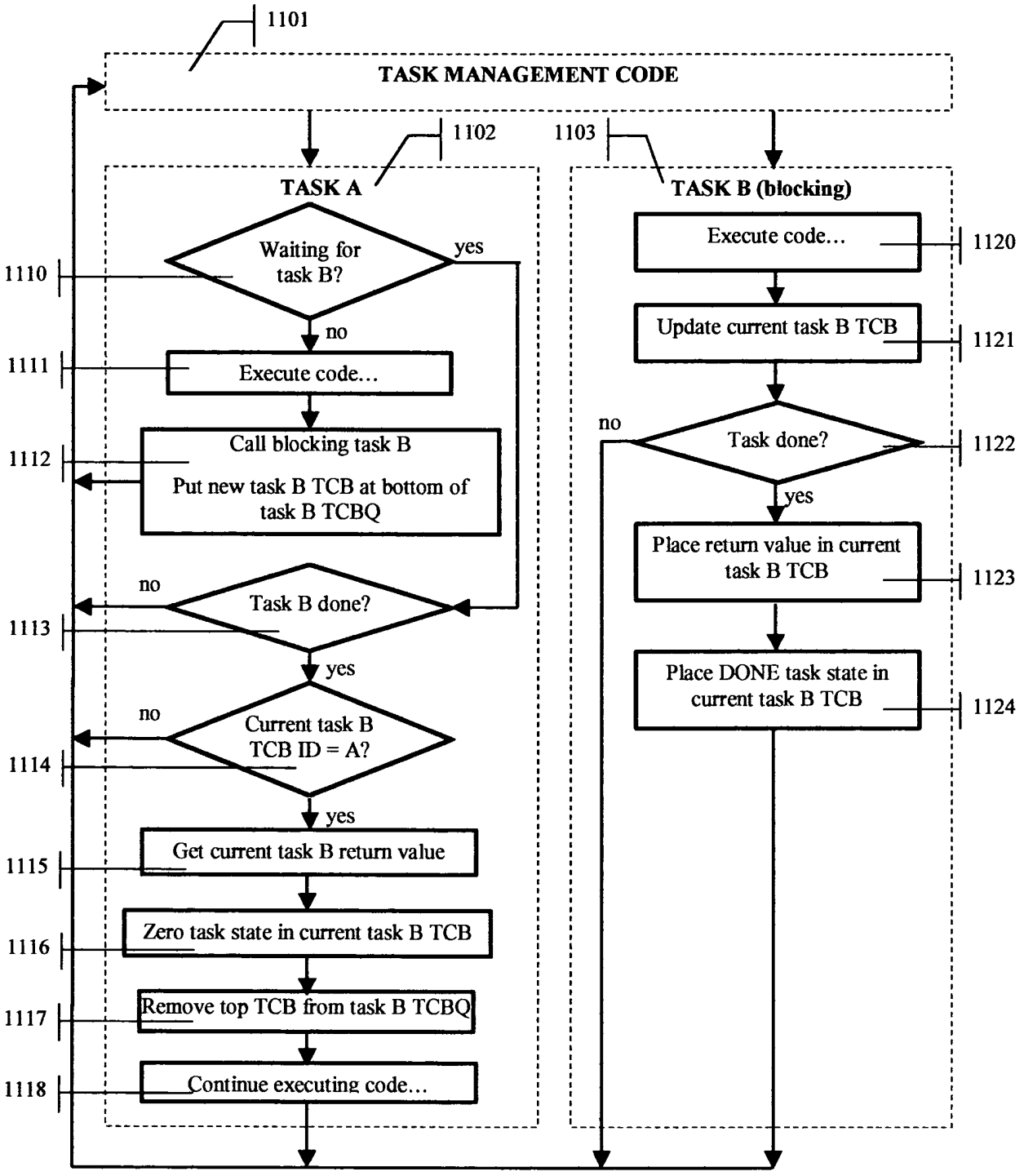
1101
TASK MANAGEMENT CODE
1102
1103
TASK A
TASK B (blocking)
Waiting for task B?
yes
1110
no
1111
Execute code...
1112
Call blocking task B
Put new task B TCB at bottom of task B TCBQ
no
Task B done?
1113
yes
no
Current task B TCB ID = A?
1114
yes
1115
Get current task B return value
1116
Zero task state in current task B TCB
1117
Remove top TCB from task B TCBQ
1118
Continue executing code...
Execute code...
1120
Update current task B TCB
1121
no
Task done?
1122
yes
Place return value in current task B TCB
1123
Place DONE task state in current task B TCB
1124

Figure 11

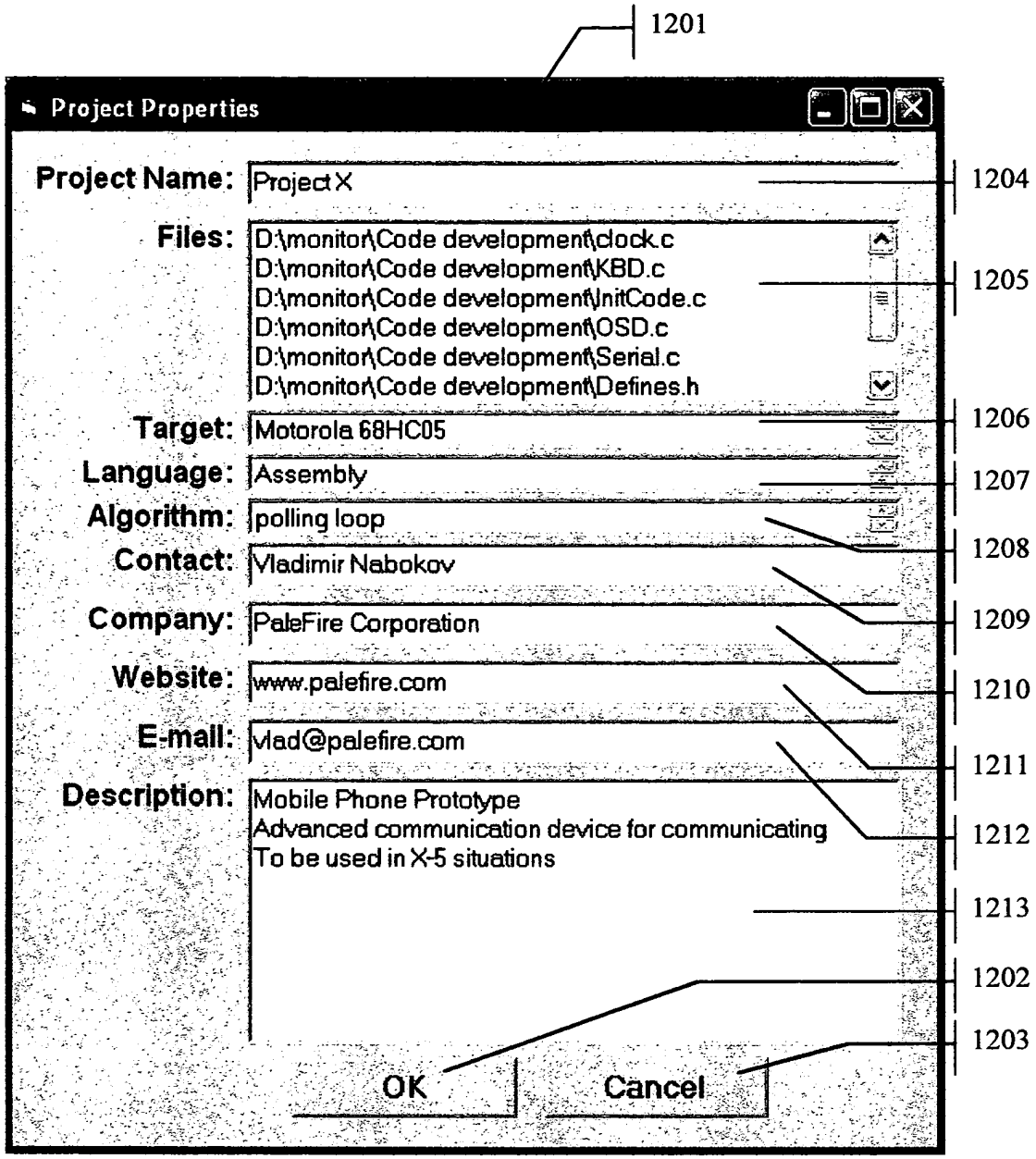
1201
Project Properties
Project Name:
Project X
1204
Files:
D:\monitor\Code development\clock.c
D:\monitor\Code development\KBD.c
D:\monitor\Code development\InitCode.c
D:\monitor\Code development\OSD.c
D:\monitor\Code development\Serial.c
D:\monitor\Code development\Defines.h
1205
Target:
Motorola 68HC05
1206
Language:
Assembly
1207
Algorithm:
polling loop
1208
Contact:
Vladimir Nabokov
Company:
PaleFire Corporation
1209
Website:
www.palefire.com
1210
E-mail:
vlad@palefire.com
1211
Description:
Mobile Phone Prototype
Advanced communication device for communicating
To be used in X-5 situations
1212
1213
1202
1203
OK
Cancel

Figure 12

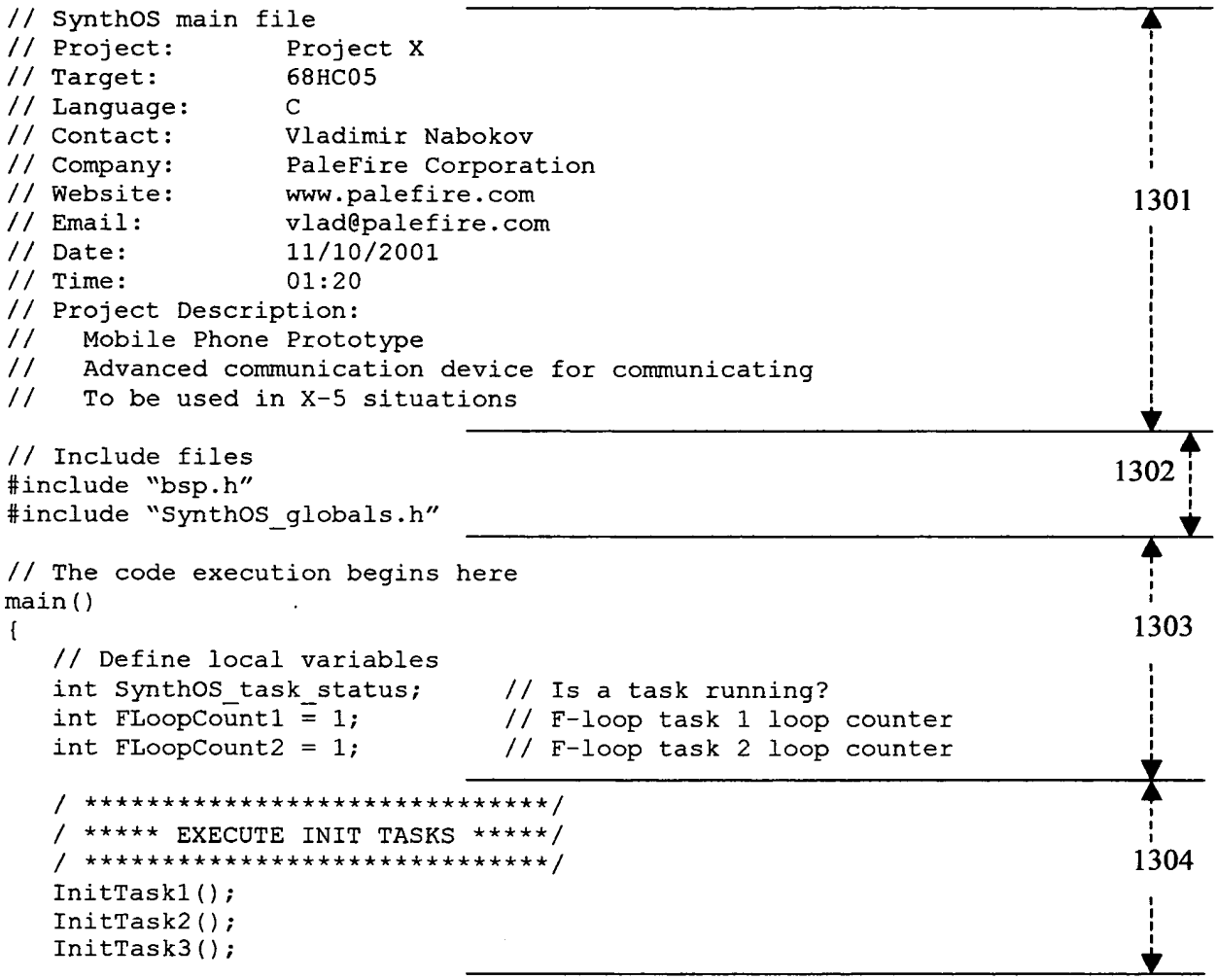
// SynthOS main file
// Project: Project X
// Target: 68HC05
// Language: C
// Contact: Vladimir Nabokov
// Company: PaleFire Corporation
// Website: www.palefire.com
// Email: vlad@palefire.com
// Date: 11/10/2001
// Time: 01:20
// Project Description:
// Mobile Phone Prototype
// Advanced communication device for communicating
// To be used in X-5 situations
1301
// Include files
#include "bsp.h"
#include "SynthOS_globals.h"
1302
// The code execution begins here
main()
{
1303
// Define local variables
int SynthOS_task_status; // Is a task running?
int FLoopCount1 = 1; // F-loop task 1 loop counter
int FLoopCount2 = 1; // F-loop task 2 loop counter
/ ******************************/
/ ***** EXECUTE INIT TASKS *****/
/ ******************************/
1304
InitTask1();
InitTask2();
InitTask3();

Figure 13a

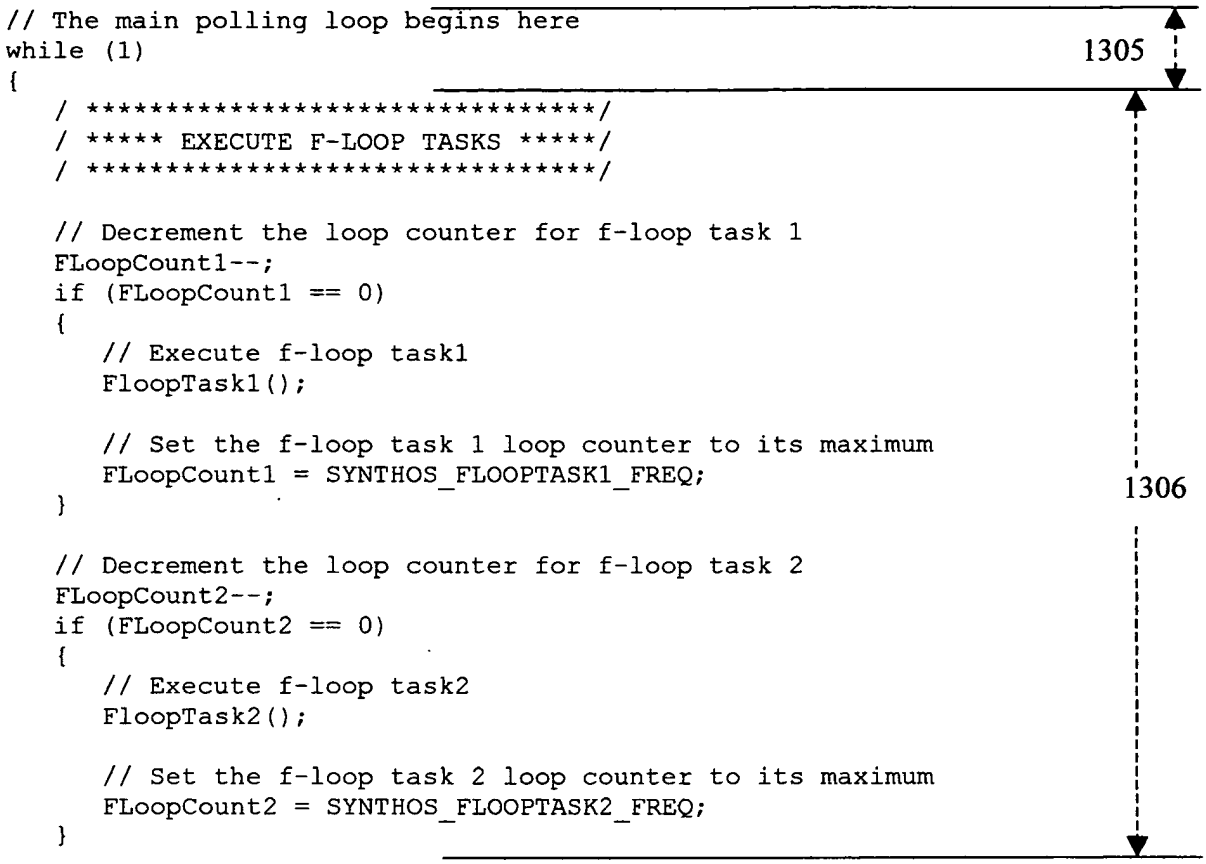
// The main polling loop begins here
while (1)
{
/ ********************************/
/ ***** EXECUTE F-LOOP TASKS *****/
/ ********************************/
// Decrement the loop counter for f-loop task 1
FLoopCount1--;
if (FLoopCount1 == 0)
{
// Execute f-loop task1
FloopTask1();
// Set the f-loop task 1 loop counter to its maximum
FLoopCount1 = SYNTHOS_FLOOPTASK1_FREQ;
}
// Decrement the loop counter for f-loop task 2
FLoopCount2--;
if (FLoopCount2 == 0)
{
// Execute f-loop task2
FloopTask2();
// Set the f-loop task 2 loop counter to its maximum
FLoopCount2 = SYNTHOS_FLOOPTASK2_FREQ;
}
1305
1306

Figure 13b

```
    / ********************************/
    / ***** EXECUTE P-LOOP TASKS *****/
    / ********************************/

    // Check status of p-loop task 1 from its TCB
    SynthOS_X_read(SynthOS_PLoopTask1_TCBQ, 0,
       SynthOS_task_status);
    // If task is not idle, execute it
    if (SynthOS_task_status != SYNTHOS_TASK_IDLE)
       PLoopTask1();

    // Check status of p-loop task 2 from its TCB
    SynthOS_X_read(SynthOS_PLoopTask2_TCBQ, 0,
       SynthOS_task_status);
    // If task is not idle, execute it
    if (SynthOS_task_status != SYNTHOS_TASK_IDLE)
       PLoopTask2();

    / ******************************/
    / ***** EXECUTE CALL TASKS *****/
    / ******************************/

    // Execute all call tasks from highest priority to lowest

    // Read the status of call task 1 from its TCB
    SynthOS_X_read(SynthOS_CallTask1_TCBQ, 0, SynthOS_task_status);
    // If task is not idle, execute it
    if (SynthOS_task_status != SYNTHOS_TASK_IDLE)
       CallTask1();

    // Read the status of call task 2 from its TCB
    SynthOS_X_read(SynthOS_CallTask2_TCBQ, 0, SynthOS_task_status);
    // If task is not idle, execute it
    if (SynthOS_task_status != SYNTHOS_TASK_IDLE)
       CallTask2();
  }
}
```

1307

1308

Figure 13c

```
// SynthOS timer ISR

#include "SynthOS_globals.h"

timer()
{
   // Decrement the p-loop task 1 counter
   PLoopTask1Counter--;
   // Check to see if it is time to execute p-loop task 1
   if (PLoopTask1Counter == 0)
   {
      // Put a new TCB in the TCBQ for task 1
      // Put task 1 into initial state 1
      SynthOS_X_write_next(SynthOS_PLoopTask1_TCBQ, 0, 1);
      SynthOS_x_write_next(SynthOS_PLoopTask1_TCBQ, 1, TIMER_ID);
      SynthOS_x_write_next(SynthOS_PLoopTask1_TCBQ, 2, a);
      SynthOS_x_write_next(SynthOS_PLoopTask1_TCBQ, 3, b);
      SynthOS_x_write_next(SynthOS_PLoopTask1_TCBQ, 4, c);

      // Set the p-loop task 1 counter to its maximum
      PLoopTask1Counter = SYNTHOS_PLOOPTASK1_PERIOD;
   }

   // Decrement the p-loop task 2 counter
   PLoopTask2Counter--;
   // Check to see if it is time to execute p-loop task 2
   if (PLoopTask2Counter == 0)
   {
      // Put a new TCB in the TCBQ for task 1
      // Put task 1 into initial state 1
      SynthOS_X_write_next(SynthOS_PLoopTask2_TCBQ, 0, 1);
      SynthOS_x_write_next(SynthOS_PLoopTask2_TCBQ, 1, TIMER_ID);

      // Set the p-loop task 2 counter to its maximum
      PLoopTask2Counter = SYNTHOS_PLOOPTASK2_PERIOD;
   }
```

1401

1402

Figure 14a

```
./ ************************************/
/ ***** EXECUTE PREEMPTIVE TASKS *****/
/ ************************************/

// Execute and pause execution of preemptive tasks

// Decrement the timer loop counter for preemptive task 1
PreemptiveTask1Counter--;
// Is it time to start executing the task?
if (PreemptiveTask1Counter == SYNTHOS_PREEMPTIVETASK1_ONTIME)
   ContextSwitchIn(PreemptiveTask1());
// Is it time to pause executing the task?
if (PreemptiveTask1Counter == SYNTHOS_PREEMPTIVETASK1_OFFTIME)
   ContextSwitchOut(PreemptiveTask1());
// When counter reaches zero, reset it to its maximum
if (PreemptiveTask1Counter == 0)
   PreemptiveTask1Counter = SYNTHOS_PREEMPTIVETASK1_MAXCOUNT;

// Decrement the timer loop counter for preemptive task 2
PreemptiveTask2Counter--;
// Is it time to start executing the task?
if (PreemptiveTask2Counter == SYNTHOS_PREEMPTIVETASK2_ONTIME)
   ContextSwitchIn(PreemptiveTask2());
// Is it time to pause executing the task?
if (PreemptiveTask2Counter == SYNTHOS_PREEMPTIVETASK2_OFFTIME)
   ContextSwitchOut(PreemptiveTask2());
// When counter reaches zero, reset it to its maximum
if (PreemptiveTask1Counter == 0)
   PreemptiveTask1Counter = SYNTHOS_PREEMPTIVETASK1_MAXCOUNT;
}
```

1403

Figure 14b